DForC: a Real-Time Method for Reaching, Tracking and Obstacle Avoidance in Humanoid Robots

I. Gori[1], U. Pattacini[1], F. Nori[1], G. Metta[1] and G. Sandini[1]

***Abstract*—We present the *Dynamic Force Field Controller* (*DForC*), a reliable and effective framework in the context of humanoid robotics for real-time reaching and tracking in presence of obstacles. It is inspired by well established works based on artificial potential fields, providing a robust basis for sidestepping a number of issues related to inverse kinematics of complex manipulators. *DForC* is composed of two layers organized in descending order of abstraction: (1) at the highest level potential fields are employed to outline on the fly collision-free trajectories that serve to drive the robot end-effector toward fixed or moving targets while accounting for obstacles; (2) at the bottom level an optimization algorithm is exploited in place of traditional techniques that resort to the Transposed or Pseudo-Inverse Jacobian, in order to deal with constraints specified in the joints space and additional conditions related to the robot structure.**

As demonstrated by experiments conducted on the iCub robot, our method reveals to be particularly flexible with respect to environmental changes allowing for a safe tracking procedure, and generating reliable paths in practically every situation.

I. Introduction

One of the most challenging tasks in robotics is concerned with the interaction of a humanoid robot with the environment, which contains interesting objects, obstacles, humans. In this scenario the robot needs to perceive its surrounding, coping with the need of properly reacting to stimuli, accounting for novel constraints and facing up to the changed conditions, while achieving its objectives.

Originally, methods for obstacle avoidance have been largely developed in the area of mobile robot navigation [1] [2] [3] [4]; dealing with the same problem in humanoid robots tends to be quite harder, since the robot structure is more complex, being equipped with a large number of variables that need to be taken into account. Therefore, as in navigation, also in the context of humanoid robots the most common approach adopted for reaching tasks while avoiding obstacles makes a systematic use of a planner [5] [6]; however it is well known that trajectory planners require a certain amount of time to compute the best trajectory. Moreover, at least in case of classical planners, the pre-computed path cannot be modified while the robot is performing its task, thus these approaches do not seem to be suitable to properly react against external stimuli.

It is worth then underlaying here two fundamental characteristics of the ideal reaching framework: first, it should constantly allow for adaptation to environmental changes, making the robot react appropriately; second, it should also work in real-time, dynamically tuning the robot's behaviors on the basis of its perception. An interesting work in this sense is the one proposed by Brock in [7] where, given a path P generated by a planner, the algorithm searches, among the set of homotopic paths with respect to P, the one that does not collide with obstacles. A controller based on potential fields is further employed to assure real-time obstacle avoidance. It employs pre-computed trajectories though, thus it involves a preliminary computation of the best path. Other approaches that have lately appeared in literature are those from Park [8], and Hoffman [9], which incorporate dynamic repulsive force fields into trajectory modeled by *Dynamic Movement Primitives* (*DMP*). The cited methods comprise dynamical models, and in this sense they share many features with the methodology proposed in this paper. Trajectories are parameterized by a (typically small) set of parameters that can be efficiently optimized with reinforcement learning methods [10]. Similar extensions can be envisaged for the technique proposed here, but these extensions definitively fall outside the current scope. Differently from the *DMP*-based methods, we avoid explicit pseudo-inverse computations relying on direct optimization approaches as described in the following sections. Another recent and interesting work in this field is [11], which mainly focuses on inverse dynamics exploiting *Quadratic Programming* (*QP*) techniques. However *QP* handles only linear constraints; for this reason, in order to solve the inverse kinematics, we resorted to *IPOPT* which is able to deal with complex non-linear formulations.

An approach based on virtual force fields for both reaching and obstacle avoidance shows to be a suitable choice; this topic has been largely analyzed in robotics by Khatib [12]. Similar properties are also shown by a more generic framework, known in literature with the name of *Passive Motion Paradigm (PMP)* [13] [14] [15], which has a relevant neurobiological background that builds on the Equilibrium Point Hypothesis developed in the 60's by Bizzi [16]. For the purpose of this paper, *PMP* can be thought of as one particular variant of the transpose Jacobian method to

Research supported by the European FP7 ICT project No. 270490 (EFAA) and project No. 270273 (Xperience).

[1]Department of Robotics, Brain and Cognitive Science (RBCS), Istituto Italiano di Tecnologia (IIT), Via Morego 30, 16163 Genova, Italy phone: +39 010 71781-420; fax: +39 010 7170-817; e-mail: name.surname@iit.it

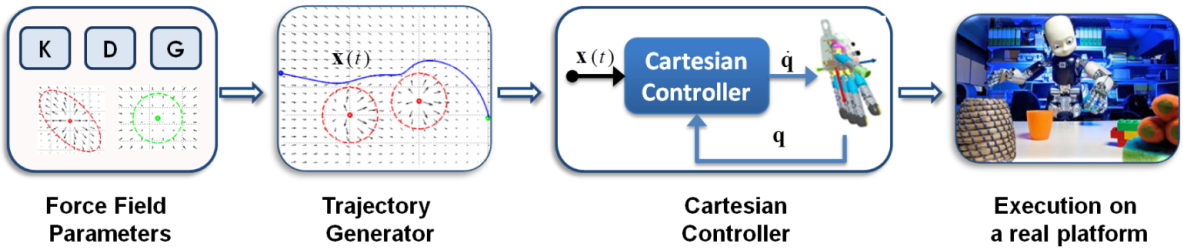

Fig. 1. System architecture. Given the force fields parameters K, D and G, a global force field is computed, and a trajectory in the task space is retrieved. Each point of the computed path is then passed to the *Cartesian Controller* that robustly copes with the inverse kinematics task and makes the robot follow the trajectory.

descend a force field.

However our main objective is to elaborate and improve the ideas behind artificial potential fields developing a system for reaching and tracking in presence of obstacles, which shows out the peculiarity of being effective, robust and real-time in order to be applied in realistic scenarios. To accomplish this task we present a two-layered framework, composed of a *Cartesian Controller* [17] at the lowest level – which solves the inverse kinematics problem and provides a reliable controller – and a force field-based trajectory generator at the highest level – which produces a collision-free path composing all the force fields representing the environment.

The remainder of this paper is organized as follows: in section II we provide a brief overview of the *PMP* framework. Section III presents a full and detailed description of our system, also highlighting innovations related to our approach. Section IV contains experimental results and finally section V completes the paper with some indications to future work.

II. Overview of the Mental Simulation in the *PMP* framework

We provide a brief overview of the *PMP* approach to make a comparison with *DForC* and to explain similarities and differences.

PMP aims to perform the *simulation of the action* in the robot beforehand, i.e. to simulate the movement taking into account additional constraints, such as joint limits and inter-limb coordination patterns. According to [14], this operation is called "mental simulation" and provides a virtual trajectory in the task space satisfying all the required objectives as well as complying with the set of constraints without executing the movement. The trajectory is generated on the basis of an attractive force field acting in the operational space exerted by a target, whilst joint limits are treated as repulsive force fields in joint space. The inverse kinematics is solved by means of the transposed Jacobian in order to map the task space force field into a torque field. *PMP* further hypothesizes that the relation between the torque field and the velocity in joint space can be modeled by an admittance matrix. Then the velocity in the Cartesian space is computed applying the differential kinematic map, and its integration finally returns the evolution of the point of the simulated trajectory over time. In order to provide a bell-shaped (i.e. human-like) velocity profile in task space and to impose a given duration to the movement, a time-varying gain function $\Gamma(t)$ is employed. The algorithm in [14] can be summarized as indicated in Tab. I.

PMP

Initialization.

- Let q be current joint configuration of the n robot's degrees of freedom.
- Let x be the corresponding end-effector pose $x = \text{kin}(q)$, embodying both position and orientation, where $\text{kin}(q)$ is the forward kinematics function.
- Let K be the stiffness of an attractive spring-mass force field.
- Let α be a n-components vector of parameters, weighting repulsive force fields that account for joint bounds.
- Let A_{int} be a *n-by-n* admittance matrix defining the correlation between the torque field and the joint velocities.
- Let $S(q)$ be a given boundary function that serves to handle joint limits.
- Let $\Gamma(t)$ a function providing a time-varying gain.
- Let x_t be the target expressed as a vector combining the desired position and orientation.
- Let τ be the specified action execution time.

Algorithm.

At each iteration step $t < \tau$ a novel point x in the Cartesian space is determined by computing the following quantities:

1) the attractive force field $F = K(x_t - x)$.
2) the torque field $T = J^T F + \alpha S(q)$.
3) joint velocities $\dot{q} = A_{\text{int}} T$.
4) velocity in the Cartesian space $\dot{x} = \Gamma(t) J \dot{q}$.
5) the new point $x(t) = \int_0^t \dot{x}(t')\, dt'$.

TABLE I
PMP algorithm [14]

III. System Overview

Our aim is to build a robust and real-time sensorimotor controller that allows the robot to accomplish on-line reaching and tracking tasks under continuously evolving environmental conditions. We are generically inspired by the theory described in [14], hence we inherit the use of force fields to describe tasks and constraints; however since the system requirements for robustness and responsiveness are central objectives, we extend the method by employing constrained optimization in order to deal with joint limits or additional

"hard" constraints of the system. We developed a two-layered architecture composed of the *Cartesian Controller* [17] at the bottom, and a force field-based trajectory generator at the top (see Fig. 1). We refer to the entire framework as *Dynamic Force Field Controller* (*DForC*). The top level computes a virtual trajectory between an initial point and the target point, taking into account the presence of obstacles, defined as objects causing obstructions; on the basis of this trajectory, the *Cartesian Controller* will assure the robot to follow the requested path. In addition, given the simplicity of the force field representation and the robustness of the constrained optimization based controller, it is easy to track a reference trajectory while avoiding collisions.

A. Cartesian Controller

The *Cartesian Controller* serves to solve the inverse kinematics problem and to guarantee that the robot actually follows the outlined trajectory: in particular it is composed of two blocks connected in cascade. The first stage is represented by the solver, whose aim is to address the inverse kinematics computation resorting to a nonlinear optimization algorithm, *IPOPT* [18]. The second module consists of a robust controller that extends the Multi-Referential Dynamical Systems approach [19]. This synchronizes two dynamical controllers, one in joint space and the other one in task space, on the basis of physiological studies [20], which assert that in humans many different reference frames are involved in the planning and control of reaching motion.

As described in [17], in the solver, we formulate the inverse kinematics problem as an optimization problem:

$$q^* = \arg\min_{q\in\mathbb{R}^n} \left(\|\alpha_t - \text{kin}_\alpha(q)\|^2 + \beta M_W(q, q_{\text{rest}})\right)$$
$$\text{s. t.} \begin{cases} \|x_t - \text{kin}_x(q)\|^2 < \varepsilon \\ q_L < q < q_U \end{cases}, \tag{1}$$

where $M_W(q, q_{\text{rest}})$ is the Mahalanobis distance:

$$M_W(q, q_{\text{rest}}) = (q_{\text{rest}} - q)^T W (q_{\text{rest}} - q). \tag{2}$$

Besides, kin_x and kin_α are the forward kinematics function that compute respectively position and orientation of the end-effector from the joint angles q, q_{rest} is a preferred joint angles which serves to bias the solution towards a resting configuration (e.g. to have the torso as close as possible to the vertical position), W is a diagonal matrix of weighting factors, β is a positive scalar weighting the influence of q_{rest}, ε is a parameter for tuning the precision of the movement, q_L, q_U represent respectively the lower and the upper bound for joint angles, α_t is the target orientation, x_t the target position and q^* is the joint configuration as solved by the optimizer. Overall, *IPOPT* takes around 20 ms to find out an appropriate solution of problem (1), thus the system can easily react in real-time, for instance in tracking scenarios. Notably in the above formulation the subtask related to the reaching in position has a higher priority with respect to the task of attaining a given orientation. Furthermore by resorting to *IPOPT*, additional constraints can be easily accounted for in the optimization problem; this way joint limits, further conditions related to the robot body structure such as arm Jacobian singularities, and even complex constraints, for instance the minimization of the energy for a certain movement, are automatically taken into account and treated with various priorities. Consequently this approach retrieves always safe configurations, while being fast enough to assure real-time performances.

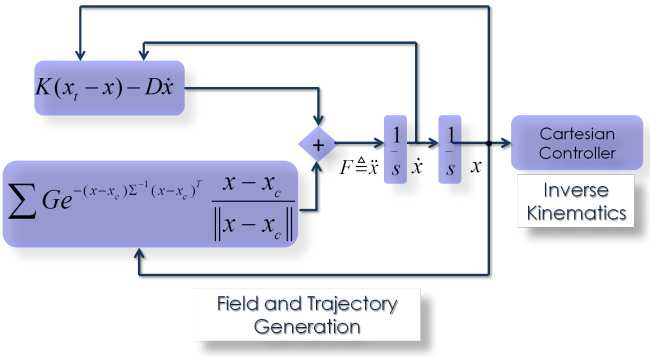

Fig. 2. Field and Trajectory Generator internal architecture. We impose that $F \triangleq \ddot{q}$ as the considered point has unit mass.

The second stage of the *Cartesian Controller* ensures the robot to follow the trajectory points and to actually reach the target in a specified amount of time. Importantly, the controller is capable of delivering proper velocity commands to the motors to achieve bell-shaped velocity profiles at the end-effector in the operational space, to drive the end-effector from the starting point to the final target resembling the human-like minimum-jerk movements. Tuning the controller responsiveness is made easy since it entails the specification of just one gain, which essentially represents the trajectory time for point-to-point movements. For additional details the reader is referred to [17].

B. Field and Trajectory Generator

The basic idea behind our force field-based trajectory generator is to build a virtual force field for each relevant object; more specifically each target is modeled as an attractive force field, whereas an obstacle generates a repulsive force field. In particular, each target generates a *spring-mass-damper* force field:

$$F_{\text{attr}}(x) = K(x_t - x) - D\dot{x}, \tag{3}$$

being x the current point of the trajectory, x_t the target point, $\dot{x}$ the current velocity of the point, K the stiffness and D the damping factor. Repulsive force fields are instead modeled as Multivariate Gaussian functions, namely

$$F_{\text{rep}}(x) = G \cdot e^{-(x-x_c)\Sigma^{-1}(x-x_c)^T} \frac{x - x_c}{\|x - x_c\|}, \tag{4}$$

where x_c is the center of the obstacle, Σ is the covariance matrix of the Gaussian function, containing information about obstacle size and pose, and G is the gain of the Gaussian

function. The combination of all the fields modeling relevant objects generates a global force field that drives the robot end-effector to the target avoiding obstacles. Specifically, at each sample time, the global force field is computed, it is integrated, and a unit mass point in the Cartesian space is obtained (see Fig. 2). This point is thus sent to the *Cartesian Controller*, so that the robot actually reaches it, and it is used to compute the successive point. The sequence of the points generated on the basis of the global force field represents the trajectory that the robot is supposed to follow. The points of the outlined trajectory are referred to as *virtual*, as they are computed only on the basis of the objects surrounding the robot; the actual end-effector trajectory will be clearly different. At each sample time, field properties can be modified, and an updated trajectory is generated on the basis of the new constraints. In this sense tracking a mobile object is straightforward, as the global force field changes on-line. Notably, tracking is performed with a sample time of $10ms$ in our architecture, and this along with the responsiveness of the inverse kinematics optimizer allows *DForC* to fully meet the real-time requirements. On the other hand the force field parameters, namely K, D and G, are not trivial to set; for instance a G much greater than K may be necessary in some cases, but it may hinder the trajectory from ending in the target position. We can get around this problem by limiting the influence of the repulsive force field to the only area occupied by the obstacle; briefly, we can cut the tails of the Multivariate Gaussian force field imposing that, if the current virtual point does not belong to the ellipsoid identified by the centroid of the obstacle and its size increased by a certain variance, the influence of the repulsive force field in that point is equal to zero:

$$\hat{F}_{\text{rep}}(x) = \begin{cases} F_{\text{rep}}, & \text{if } x \in ell(x_c, \Sigma) \\ 0, & \text{otherwise} \end{cases}; \tag{5}$$

$ell(x_c, \Sigma)$ identifies the ellipsoid described by the centroid x_c and the covariance matrix Σ. This solution turns to be simple, while preventing the end-effector from getting stuck in local minima as experimentally verified in the majority of the cases that appear in real scenarios. Nonetheless, in the future we plan to integrate a more sophisticated formulation that ensures a complete protection from the local minima phenomenon [3]. Modeling the obstacles with Multivariate Gaussians without tails implies that the exerted force is discontinuous at the boundaries of obstacles; nevertheless this discontinuity does not affect the final smoothness of the end-effector trajectory thanks to the decoupling between the virtual target point and the intrinsic Cartesian dynamic as enforced by the controller.

C. Advances

For our *Dynamic Force Field Controller* (*DForC*), we took inspiration from the work in [14], which provides theory and description a of the *PMP* method; here we comment further on the differences between the two methods with the aim of providing a comparison.

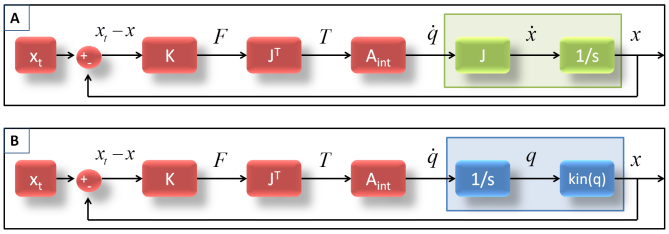

Fig. 3. A: the model employed by *PMP*, where the end-effector trajectory is computed by integrating the velocity in the task space causing drift phenomena. B: an efficient alternative model resorting to the integration of joints velocities.

*1) **Actual Robot Movements**:* according to [14], the trajectory $x(t)$ is retrieved by integrating the fundamental differential kinematic map (see Fig. 3-A)

$$x(t) = \int_0^t \dot{x}(t')\, dt' = \int_0^t J\dot{q}(t')\, dt'. \tag{6}$$

This formulation does not provide the regulator needed to control the robot, since it deals only with the simulated trajectory in Cartesian space. On the contrary, the robot is typically controlled in joint space as for example with velocity profiles in closed loop with respect to joint angles feedback, with the goal of compensating for the deviations of the real system from the ideal integrator (e.g. because of dynamics). Therefore, in order to analyze the actual behavior of the end-effector on the basis of the $\dot{q}$ computed via the algorithm in Table I, it is necessary to calculate $x(t)$ as follows:

$$x(t) = \text{kin}\left(\int_0^t \dot{q}(t')\, dt'\right). \tag{7}$$

Even though Eq. (6) is mathematically equivalent to Eq. (7) (the proof is out of the scope of this paper), the numerical integration in the first leads to drift (see [21]); as result the end-effector pose computed via Eq. (7) is different from the pose computed by purely integrating the velocity in the Cartesian space (Fig. 4). Hence, the diagram depicted in Fig. 3-B that implements Eq. (7) gives account of a possible law for a controller that, on one hand, preserves the formulation described in [14] and, on the other hand, can be conveniently plugged into a canonical closed loop schema and used for our experiments.

Differently in our method, we compute the virtual trajectory in the Cartesian space at top level without involving transformation in the joint space nor resorting to the inverse kinematics; afterward, the *Cartesian Controller* takes care of calculating the most suitable joint position for a certain point in task space, and guarantees that the robot reaches it employing a human-like trajectory profile.

*2) **Facing basic constraints**:* the *PMP* approach treats basic constraints, such as joint limits, as repulsive force fields that will be added to the main torque field, namely

$$T = J^T F + \alpha S(q), \tag{8}$$

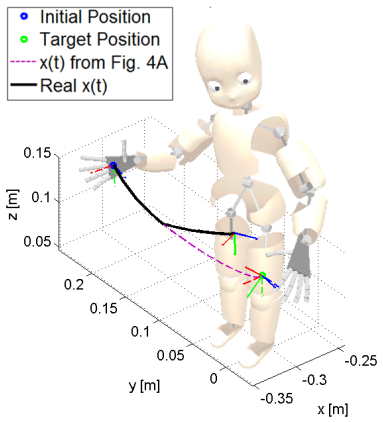

Fig. 4. Drift phenomena due to the numerical integration of $\dot{x}$: the purple trajectory is the one computed by system represented in Fig. 3-A, we then integrate $\dot{q}$ obtaining q, and we compute the real x by direct kinematics; the resulting trajectory is the black one. The blue point is the initial position, and the green one is the target position. iCub has been sketched for sake of clarity, even though its dimensions do not correspond to the axes units.

where $S(q)$ is a vector composed of "rotated" sigmoids S_i:

$$\begin{cases} \lambda_i = \dfrac{q_i - q_i^{min}}{q_i^{max} - q_i^{min}} - 1 \\ S_i = \log \dfrac{1 - \lambda_i}{1 + \lambda_i} \end{cases}, \tag{9}$$

aimed at pushing away the joint angles from their respective joint limits, and α is a vector of parameters that weights the repulsive force fields on each joint. This approach may sometimes prevent the trajectory from reaching a target located at the extremity of the robot's workspace. Depending on the value of α, this formulation may as well not reach the target. Typically, the *PMP* framework is not concerned with other types of constraints, as for instance, those related to fragile points that are specific to the robot's structure (e.g. the constraints induced by the tendons lengths in the iCub [22]). On the contrary, as stated in Eq. (1), we treat joint limits as higher priority constraints in the inverse kinematics formulation; indeed, we require the solution q^* to lie between lower and upper bounds (q_L, q_U) of physically admissible values. Thus, we can guarantee safe configurations while following the demanded path.

3) ***Handling Time Execution****:* as described in [14] *PMP* requires to fix the duration of the action execution τ, which then determines the function $\Gamma(t)$, namely

$$\begin{cases} \Gamma(t) = \dfrac{\dot{\xi}}{1 + \varepsilon - \xi} \\ \xi(t) = 6(t/\tau)^5 - 15(t/\tau)^4 + 10(t/\tau)^3 \end{cases}, \tag{10}$$

where the small constant ε is needed to prevent the function from diverging toward infinity as t approaches τ. The $\Gamma(t)$ time-varying gain function acts as a time base to complete the movement in τ seconds and therefore it replicates a bell-shaped velocity profile of the end-effector. We observed though that this can lead to other issues as for example preventing the trajectory from ending at the target position, especially when a joint approaches a workspace limit.

This does not happen in *DForC*, in fact the low-level controller guarantees that the target is reached in finite-time, as long as the target configuration is geometrically meaningful.

4) ***Orientation****:* traditionally, when a humanoid robot performs a reaching task, it is crucial to cope with circumstances when the final object is attainable in position and thus can be touched, but the orientation cannot be reached perfectly at the same time. This happens because the ultimate goal of reaching is deemed to be a successful grasping action or more in general objects manipulation; to this end, different priorities can be conveniently assigned to reaching in position and reaching in orientation. *PMP* does not take this issue into account, in fact it handles constraints on final orientation and final position with the same priority.

On the contrary, as remarked in Eq. (1), the *Cartesian Controller* treats the orientation as a lower priority subtask; specifically, when the robot is asked to perform a reaching task, it tries to attain the target with the requested orientation, but if the desired configuration is not feasible the *Cartesian Controller* just drives the end-effector to the demanded position without imposing to perfectly reach the orientation at the same time. This method is effective in most of the cases, and it is proved (see [17] [18]) that it outperforms even classic methods based on the *homogeneous solution* ([23]), which exploit the Jacobian null-space in order to deal with secondary tasks (e.g. joint limits avoidance). In summary, we do not control the orientation during the obstacle avoidance planning but only during the *Cartesian Controller* execution. Nevertheless we are currently working on defining further force fields in our trajectory generator that will affect the hand rotation.

IV. Experiments

The experiments presented in this paper have been performed on the iCub [24], a 53 degrees of freedom humanoid robot, developed by the RobotCub project. The iCub is equipped with cameras, proprioception, force-torque sensors at the limbs and artificial skin. Furthermore, from the motor control point of view, the iCub can be controlled in different modalities according to a set of different controllers selected at run time: a desired set-point for the joint can be thus achieved with precomputed minimum-jerk trajectories or by employing custom control laws that rely on velocity commands. Torque and consequently impedance control are also available. In the context of these experiments we let the iCub perform reaching and tracking by employing 10 DOF in total, considering the 3 DOF of the torso along with the 7 DOF of the arm.

Experiments conducted in parallel on both *PMP* and *DForC* are presented to highlight the improved behavior of the proposed framework. We test the *PMP* mental simulation in three cases: first we set a simple collision-free scenario,

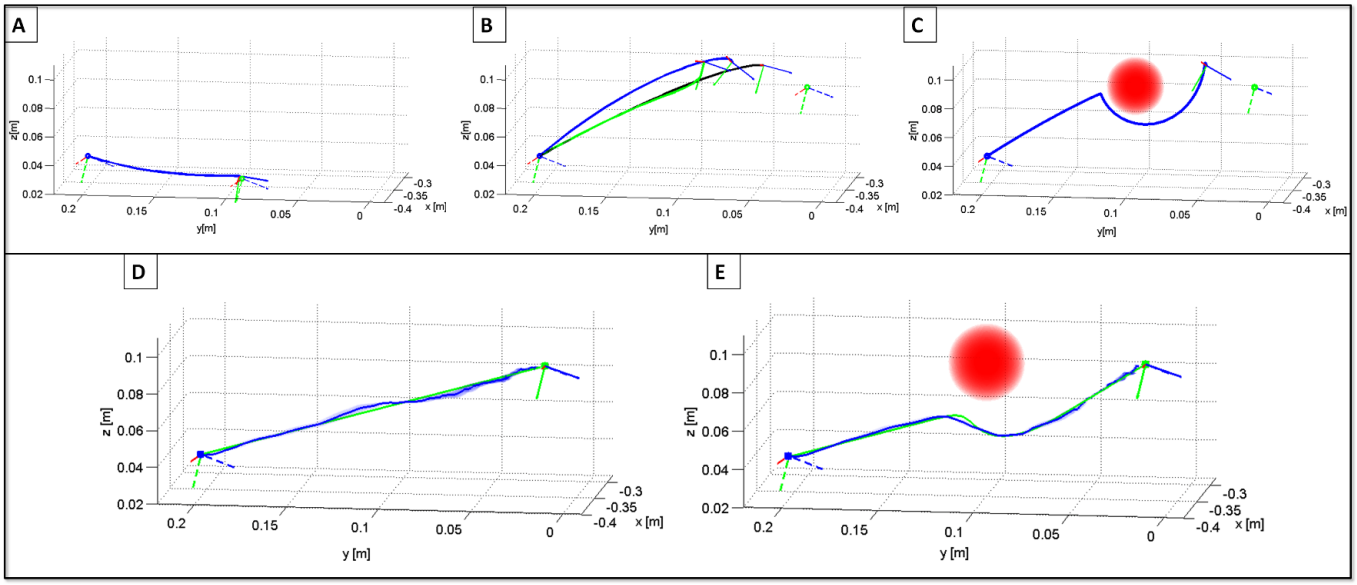

Fig. 5. In the first row experiments to assess the *PMP* framework in reaching tasks are depicted. Figure A shows that in simple conditions *PMP* obtains good performances: the last point of the trajectory is only $4\ mm$ far from the target. Figure B shows how repulsive force fields in the joint space tend to be inadequate to model joint limits as the target remains about $3\ cm$ far from the last point of the trajectory (blue line). Also the $\Gamma(t)$ function, in the specified scenarios, entails the target to remain unattained for about $5\ cm$ (black line). Finally, the control of the orientation was enabled as well (green line), and figure B shows that the trajectory diverges. Figure C reports *PMP* behavior in presence of obstacles: collisions are avoided, but the previously described issues hold. The second row shows experiments conducted on *DForC* in reaching tasks when both the position and the orientation are controlled. Figure D reports that the target is reached by both the virtual trajectory (green line) and the path actually followed by the robot (blue line). In figure E reaching in presence of an obstacle is analyzed: the obstacle is successfully bypassed by both the virtual and the real trajectories.

then we place the target at the extremity of the iCub's workspace, and finally we stress the previous setting adding an obstacle. Our framework is then assessed in the same scenarios. Additionally, we also prove that our system easily accomplishes tracking tasks while avoiding collisions and, specifically, we carry out an experiment where the iCub is supposed to follow a moving target while an obstacle suddenly appears in the scene trying to occlude the target.

A. Reaching Performances

As we mentioned in Sec. III-C, we employed the model showed in Fig. 3-B in order to avoid numerical integration of velocities. We empirically explored the space of the parameters trying to achieve the best response in terms of final position errors, and we ended up choosing α equal to a vector of $0.05\ Nm$, $K = 4N$ and the diagonal of the admittance matrix A_{int} filled with values in the range $(0, 1]\ rad/(Nm \cdot s)$. Fig. 5-A shows that, where no joint limits are pushed toward their bounds and the orientation is not controlled, *PMP* achieves good performances, and the $\Gamma(t)$ function appears to play its role (the target is reached in $\tau = 3\ s$). Conversely, Fig. 5-B highlights how, with the target located at the extremity of the robot's workspace, the goal is not reached. The blue line reports the case where the repulsive force fields acting on the joints, as proposed by the *PMP*, are enabled: notably, since the reaching of the target requires a joint of the wrist to push against its upper limit, the repulsive force fields on the joints prevents the trajectory from attaining the target. The black line sketches out the case where $\Gamma(t)$ is active and one joint of the iCub's wrist is pushed toward its upper limit, being repulsive force fields on joint limits disabled: when there are slowdowns in the movements due to joints strongly pushing toward the bounds, the $\Gamma(t)$ function no longer guarantees that the action is completed in a finite time. Remarkably, in both cases the orientation is not controlled. By contrast, the green line reports the case where the $\Gamma(t)$ is disabled and the orientation is controlled. We remarked in Sec. III-C how the reaching in position is largely influenced by the reaching in orientation, and in fact the green trajectory shows to diverge from the target. Finally we imposed a spherical obstacle modeled as described in Sec. III-B, where the covariance matrix Σ in Eq. (5) has non-zero elements only on the diagonal whose values are put equal to the radius of the obstacle (i.e. $r = 2\ cm$) and $G = 8\ N$; Fig. 5-C highlights that force fields on obstacles in the task space appear to be effective since the obstacle is successfully bypassed, although the previously described drawbacks hold and in fact the target is not attained. Furthermore the tuning of the parameters is crucial.

We evaluated then *DForC* in the same configuration settings: the initial position and the target points are kept unchanged, and we asked the system to generate the virtual trajectory and to actually execute the reaching task controlling both position and orientation. We keep the same parameters for K, G, r and Σ, and we set $D = 8\ Ns/m$. We

performed this experiment 10 times, obtaining the average path displayed in Fig. 5-D as a blue line, with a variance represented by the shadowed area around the main path; the virtual trajectory is the continuous line in green. In our case, the green trajectory has a different meaning with respect to the blue one; the higher level of *DForC* is in charge of generating the green path – which is supposed to be collision-free – only on the basis of the force fields exerted by the target and the objects surrounding the robot. Thus, it does not take into account constraints on joints or on the robot's structure, or on orientation, and not even on time execution. Conversely, the blue path represents the actual end-effector movement as produced by the *Cartesian Controller* whose main goal is to make the robot track the virtual trajectory.

We tested our system also in a reaching scenario where obstacles are present. We conducted the experiment 10 times with the results displayed in Fig. 5-E: the blue line represents the average path with a variance reported as the shadowed area around, the red ball represents the obstacle, and the virtual trajectory is colored in green. The plot clearly shows how both the green and the blue trajectories bypass the obstacle. Notably, the measure of variance of the trajectories in both the experiments on *DForC* is significantly small, entailing a high repeatability of the tasks. Furthermore, we remark that with our formulation the joint limits are not exceeded, the action is completed in a finite time and the desired orientation is attained without disrupting the reaching in position.

B. Tracking Performances

As *DForC* allows tracking tasks to be easily fulfilled, we present a simple demonstration where the iCub robot has to follow a green ball while avoiding an obstacle represented by a red ball. The two balls can be freely moved in space by the experimenter and then tracked by a method based on particle filtering [25] that exploits color and 3D shape information. Both balls, when in the robot's range of vision, are tracked, and their Cartesian positions are continuously sent to the controller, which consequently updates the field generation. We set the field parameters K, D and G respectively to $4\ N/m$, $8\ Ns/m$ and $10\ N$, whereas the covariance matrix Σ is diagonal with elements equal to the radius of the red ball (i.e. $5\ cm$). Finally, the parameter T, determining the robot's responsiveness in executing a point-to-point trajectory, amounts to $0.6\ s$. Fig. 6 shows the results of the proposed experiment: on the left the iCub is depicted while it is following the green ball avoiding the red one, whereas the middle image gives a pictorial representation of the iCub as seen within the iCubGui, a graphical user interface that constantly updates the full state of the robot; it also displays the perceived objects as colored cuboids, and it allows drawing simple paths. In this snapshot, the iCub has just avoided the obstacle (big red cuboid) – which keeps following a descending trajectory – and it is approaching the target (small green cuboid). On the right side of Fig. 6 four trajectories are shown: the red one depicts the descending obstacle, the green one represents the target, the cyan and the blue paths are respectively the virtual trajectory as generated by the framework and the actual path executed by the robot's end-effector. Notably, the locations of the red and the green balls are affected by a relevant noise, which stems from the measurements obtained with the particle filter. However, despite the noise in the perception, the generated trajectories remain remarkably smooth thanks to the intrinsic dynamics of the system, and thus allow the robot to attain the target while avoiding the moving obstacle. It is also worth noticing that target and obstacle move continuously (every $10ms$ the fields are updated accounting for object position changes), and still our framework is able to handle sudden and unexpected deviations in real-time. In order to better clarify the evolution of this experiment, we highlight two specific situations, by capturing two intermediate positions of all the involved objects: the situation referred as number 1 in Fig. 6 (right) represents the positions of the green ball, the red ball, the end-effector and the virtual point, when the obstacle has not influenced the end-effector's movement yet. Further, the situation referred as number 2 has been caught in a subsequent instant, when the end-effector has just avoided the obstacle, and the target is already in its final position. The red and green solid balls account for the obstacle and the target, respectively, when in final positions. From the described figure it is possible to infer that our *DForC* framework can easily make the robot accomplish tracking tasks, robustly following an outlined trajectory while avoiding obstacles, and it is capable of handling intrinsically dynamic scenarios that vary in real-time.

V. Conclusion and Future Work

A two-layered framework called *Dynamic Force Field Controller* (*DForC*) is proposed; it aims at effectively controlling the movement of a humanoid robot in complex reaching and tracking tasks in the presence of obstacles and in real-time, handling also sudden and unexpected deviations of the objects. Inspired by theories on the artificial potential fields, the high level layer is designed to include a trajectory generator that computes a collision-free path for the end-effector, connecting the starting point and the desired location in the operational space by assigning attractive or repulsive virtual force fields to obstacles and targets, respectively. Then, a robust *Cartesian Controller* running at a lower level is responsible for reliably solving inverse kinematics through nonlinear constrained optimization that is proved thus to be fast enough for real-time computation. Importantly, the entire framework abstracts away the platform: even though we carried out our tests on the iCub, the system can be ported to different robots, as the trajectory generator is a completely hardware-independent component, whereas the *Cartesian Controller* is a general-purpose tool that can be easily adapted to the user needs requiring only the knowledge of the manipulator kinematics.

Our future work is directed at incorporating a body-schema and therefore to account for self-collision avoidance

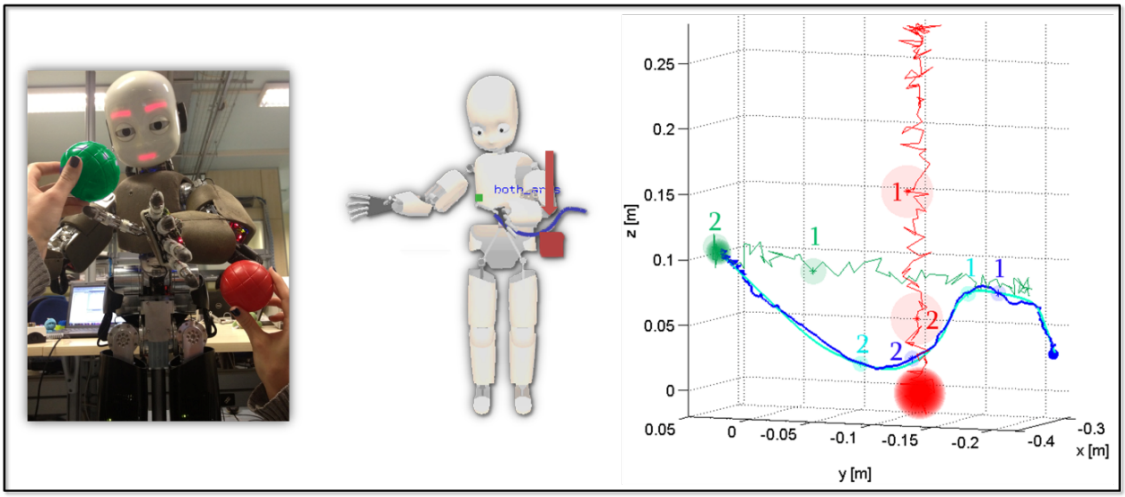

Fig. 6. Tracking experiment. On the left, the iCub while it is following the green ball avoiding the red one. On the middle, the iCubGui pictorially representing the objects perceived by the robot and depicting the trajectory that has just been executed. On the right, the trajectories of the red ball, the green ball, the end-effector and the virtual point generated by the up level of *DForC*. Two instants of the evolution are captured, and they are represented with the transparent balls depicted in the figure. The green and the red solid balls represent, respectively, the target and the obstacle in their final position.

or bimanual manipulation. To this end, our goal is to exploit the iCub tactile sensors recently completed.

References

[1] V. J. Lumelsky and A. A. Stepanov, "Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape," *ALGORITHMICA*, 1987.

[2] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," in *IEEE Transactions on Systems, Man, and Cybernetics*, 1989, pp. 1179–1187.

[3] M. Randazzo, A. Sgorbissa, and R. Zaccaria, "Nav: Navigation without localization," in *International Conference on Intelligent Robots and Systems*, 2006.

[4] M. Zucker, J. J. J. Kiffner, and M. Branicky, "Multipartite rrts for rapid replanning in dynamic environments," in *ICRA*, 2007.

[5] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, "Motion planning for humanoid robots under obstacle and dynamic balance constraints," in *in IEEE Int. Conf. on Robotics and Automation, 2001*, 2001, pp. 692–698.

[6] D. Bertram, J. J. J. Kiffner, R. Dillmann, and T. Sdfour, "An integrated approach to inverse kinematics and path planning for redundant manipulators," in *ICRA*, 2006.

[7] O. Brock, O. Khatib, and S. Viji, "Task-consistent obstacle avoidance and motion behavior for mobile manipulation," in *In Proceedings of IEEE International Conference on Robotics and Automation*, 2002, pp. 388–393.

[8] D. H. Park, H. Hoffmann, P. Pastor, and S. Schaal, "Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields," in *International Conference on Humanoid Robots*, 2008.

[9] H. Hoffmann, P. Pastor, D. H. Park, and S. Schaal, "Biologically-inspired dynamical systems for movement generation: Automatic real-time goal adaptation and obstacle avoidance," in *ICRA*, 2009, pp. 2587–2592.

[10] J. Peters, K. Mülling, J. Kober, D. Nguyen-Tuong, and O. Krömer, "Towards motor skill learning for robotics," *Proceedings of the International Symposium on Robotics Research ISRR*, pp. 1–14, 2009. [Online]. Available: http://www.kyb.mpg.de/publications/attachments/ISRR2009-Peters_6069[0].pdf

[11] N. Mansard, "A dedicated solver for fast operational-space inverse dynamics," in *ICRA*, 2012.

[12] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Rob. Res.*, vol. 5, 1986.

[13] F. A. Mussa Ivaldi, P. Morasso, and R. Zaccaria, "Kinematic networks. a distributed model for representing and regularizing motor redundancy," in *Biological Cybernetics*, 1988, pp. 1–16.

[14] V. Mohan, P. Morasso, G. Metta, and G. Sandini, "A biomimetic, force-field based computational model for motion planning and bimanual coordination in humanoid robots," *Auton. Robots*, vol. 27, pp. 291–307, 2009.

[15] V. Mohan and P. Morasso, "Passive motion paradigm: an alternative to optimal control," *Frontiers in Neurorobotics*, vol. 5, 2011.

[16] E. Bizzi, F. A. Mussa Ivaldi, and S. Giszter, "Computations underlying the execution of movement: A biological perspective." in *Science*, 1991, pp. 287–291.

[17] U. Pattacini, F. Nori, L. Natale, G. Metta, and S. G., "An experimental evaluation of a novel minimum-jerk cartesian controller for humanoid robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 1668–1674.

[18] A. Wächter and L. Bielger, "On the implementation of a primaldual interior point filter line search algorithm for large-scale nonlinear programming," in *Mathematical Programming*, 2006, pp. 25–57.

[19] H. Hersch and A. G. Billard, "Reaching with multi-referential dynamical systems," in *Autonomous Robots*, 2008, pp. 71–83.

[20] J. E. Paillard, *Brain and Space*. London: Oxford University Press. Chaps. from Arbib, Berthoz and Paillard, 1991.

[21] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 2008, ch. 3(7).

[22] A. Parmiggiani, M. Randazzo, L. Natale, G. Metta, and G. Sandini, "Joint torque sensing for the upper-body of the icub humanoid robot," in *IEEE International Conference on Humanoids Robots*, 2009.

[23] O. Khatib, L. Sentis, and J. Park, "A unified framework for whole-body humanoid robot control with multiple constraints and contacts," in *Proc. of the European Robotics Symposium*, 2008, pp. 303–312.

[24] G. Metta, G. Sandini, D. Vernon, L. Natale, and F. Nori, "The icub humanoid robot: an open platform for research in embodied cognition," in *8th Workshop on Performance Metrics for Intelligent Systems*, 2008.

[25] M. Taiana, J. Santos, J. Gaspar, J. Nascimento, A. Bernardino, and P. Lima, "Tracking objects with generic calibrated sensors: an algorithm based on color and 3d shape features," in *Robotics and Autonomous Systems, special issue on Omnidirectional Robot Vision*, 2010, pp. 784–795.